LOGO: A New Distributed Leader Election Algorithm in WSNs with Low Energy Consumption

Ahcène Bounceur, Madani Bezoui, Umber Noreen, Reinhardt Euler, Farid Lalem, Mohammad Hammoudeh, Sohail Jabbar

LOGO: A New Distributed Leader Election Algorithm in WSNs with Low Energy Consumption

Ahcène Bounceur[1], Madani Bezoui[2], Umber Noreen[1], Reinhardt Euler[1]
Farid Lalem[1], Mohammad Hammoudeh[3], and Sohail Jabbar[4]

[1] Université de Bretagne Occidentale
CNRS Lab-STICC Laboratory, UMR 6285
Brest, France,
Ahcene.Bounceur@univ-brest.fr
[2] Université de Boumerdes,
Boumerdes, Algerie
[3] University of Manchester,
Manchester, UK
[4] Department of Computer Science
National Textile University
Faisalabad, Pakistan

Abstract. The Leader Election Algorithm is used to select a specific node in distributed systems. In the case of Wireless Sensor Networks, this node can be the one having the maximum energy, the one situated on the extreme left in a given area or the one having the maximum identifier. A node situated on the extreme left, for instance, can be used to find the boundary nodes of a network embedded in the plane. The classical algorithm allowing to find such a node is called the Minimum Finding Algorithm. In this algorithm, each node sends its value in a broadcast mode each time a better value is received. This process is very energy consuming and not reliable since it may be subject to an important number of collisions and lost messages. In this paper, we propose a new algorithm called LOGO (Local Optima to Global Optimum) where some local leaders will send a message to a given node, which will designate the global leader. This process is more reliable since broadcast messages are sent only twice by each node, and the other communications are based on a direct sending. The obtained results show that the proposed algorithm reduces the energy consumption with rates that can exceed 95% compared with the classical Minimum Finding Algorithm.

Key words: Wireless Sensor Network, Leader Election, Distributed algorithms

1 Introduction and related work

Wireless Sensor Networks (WSNs) are useful in situations where we need to measure environmental data, especially when the measures must be taken in

dangerous or inaccessible places. In the context of Internet of Things (IoTs) and Smart-Cities, WSNs can be used to detect free places in car parkings, to secure and detect intrusions around sensitive sites, to predict and detect fires, etc. They are composed of autonomous sensor nodes that communicate between them using short wireless communication in order to exchange messages and data. They can also communicate with a static or mobile base station in order to transmit the collected data.

The main context of this paper, is the surveillance of sensitive and dangerous sites where one needs to find the boundary nodes of a WSN. A recent algorithm, called D-LPCN [1] (Distributed Least Polar-angle Connected Node), can be used for this purpose. The nodes must communicate with each other. This characteristic is necessary in order to be able to detect faulty nodes [2]. These algorithms start from the node which is on the extreme left of the network. To find this particular node, one can use any Leader Election algorithm which can also be used for other applications and actions, like for example, coordination, cooperation, etc. Leader election is a complex problem in distributed systems since the data are distributed among the different nodes, which are geographically separated as is the case for WSNs. Several approaches are available to deal with this problem. The Minimum Finding Algorithm [3] is the classical one and it is based on updating and broadcasting each smallest received value. A new leader selection algorithm for homogenous wireless sensor networks is presented in [4]. In [6], an improved version of the well-known Leader Election Ring algorithm is presented, where the authors to reduce the number of election messages by making assumptions on perfect clock synchronization and a perfect connection between transmitter and receiver. These assumptions are not realistic in wireless sensor networks and they require additional complex algorithms to deal with this synchronization. In [7], another improved version of Bully's algorithm is described, which acquires a smaller number of transmissions for leader election but takes more time. During leader election, a node will compare its value with the received value and only transmit the greater one. In [8], the network is divided and a pre-election to select a provisional leader is performed. The main drawback of this approach is when a node crashes and the contents of the memory will be completely removed. In [9], the number of nodes which can detect the failure is bounded before starting the Leader election algorithm. In this paper, the algorithm's time and complexity remain optimal even in worst case scenarios. The authors of [10] have modified Bully's algorithm in a way to improve the processing time. Their main contribution is that the election of a node is done on the basis of its performance and operation rate instead of on its higher identifier. Their algorithm allows to determine a leader before an existing leader dies. As in [6], the authors of [11] made the assumption of a perfect transmission connection and the time on air and collision scenarios is ignored, while implementing fault detection algorithms. Also, a series of dynamic leader election protocols in broadcast networks has been proposed. In [12], it is suggested to choose one leader and one leader assistant, so that in the case where the leader node crashes, the assistant node can take over the charge and coor-

dinate other nodes. This can significantly decrease the total number of elections in the network, especially when the network size is large. A similar approach is followed in [5] whose approach does not rely on any particular network topology. In [13], two main algorithms are presented. The Bully algorithm [17] and the Ring Algorithm [14]. In [15], the author has proposed two algorithms working in the case of asynchronous networks. Both of the proposed algorithms can reduce the time complexity.

Except for the Minimum Finding Algorithm [3] and the algorithm of [15], the presented algorithms can be used only in the case of synchronous networks. In this article, we present a new algorithm that works with asynchronous networks without making any assumption on the topology. This algorithm has been compared with the classical Minimum Finding Algorithm, and the simulation results have shown that its complexity regarding exchanged messages is reduced by rates that can exceed 95%.

The remainder of the paper is organized as follows: In the following section, the Minimum Finding Algorithm will be reviewed. In Section 3, the Local Minimum Finding Algorithm (LMF) will be presented. Section 4 will present the proposed approach. The platform CupCarbon, which is used to implement the proposed algorithm, will be described in Section 5. In Section 6, simulation results will be presented. Finally, Section 7 concludes the paper.

2 The Minimum Finding Algorithm

In this section, we will present an algorithm that allows to determine a node leader representing the node with minimum or maximum value v. This value can represent the battery level, the residual energy, the identifier, the local energy, the x-coordinate in a network, etc. This algorithm is based on the Minimum Finding Algorithm presented in [3, 16] which itself relies on the tree-based broadcast algorithm. The same algorithm can be used to find the maximum value. It can be described as follows. At the beginning, each node of the network assumes that its local value is the minimum of the network (the leader) and assigns it to the variable x_{min}. This value will be broadcasted and the corresponding node will wait for incoming x_{min} values from its neighbors. If a received value x_{min} is less than its local x_{min} value then this one will be updated and broadcasted again. This process is done repeatedly by each node as long as a received value is less than its local x_{min} value. After a given time t_{max}, only the leader, with the smallest value, will not receive a value that is smaller than its local x_{min} value.

Algorithm 1 is the pseudo-code of this process, where t_0 is the time of the first execution of the algorithm, that can correspond to the first powering-on of a sensor node, t_c the current local time of a sensor node, and t_{max} the maximally tolerated running time of the algorithm from the first execution to the current time of a sensor node.

Algorithm 1 *MinFind*: The pseudo-code of the classical Leader Election Algorithm

Input: t_{max}, v
Output: leader

```
 1: leader = true;
 2: t_0 = getCurrentTime();
 3: x_min = v;
 4: send(x_min, *);
 5: repeat
 6:     x = read();
 7:     if (x < x_min) then
 8:         leader = false;
 9:         x_min = x;
10:         send(x_min, *);
11:     end if
12:     t_c = getCurrentTime();
13: until (t_c − t_0 > t_max)
```

In order to set the value of t_{max}, one needs to calculate the time complexity of this algorithm. For this purpose, let us consider the worst case represented by a linear network with n nodes, where we are searching for the node with minimum x-coordinate. This node, situated on the extreme left, will send only 1 message and will receive only 1 message. Nevertheless, the right-most node will receive and send $n-1$ messages of the received assumed x_{min} coordinate, since it is the node having the largest x-coordinate. Therefore, each one of the other nodes, except the extreme left one, has at least one node on its left. Thus, these nodes will broadcast the newly received x_{min}.

Altogether, the message complexity is equal to $M[MinFind] = 2(n-1) = 2n-2$. If we assume that a sensor node can send and receive messages simultaneously (full-duplex communication) the overall time complexity $T[MinFind] = n-1$. Since the time complexity is known, it is possible to estimate the value of t_{max}, the required time to find the leader. For example, in a network of 100 sensor nodes, with 1024 bits message size sampled with a 250 kb/s frequency (802.15.4 standard based network), 406 ms are required to find the leader. In this article, we have simulated two networks of 100 sensor nodes using the CupCarbon simulator. The first network is assumed to be linear (cf. Figure 1) and second is assumed to be random (cf. Figure 2). The simulation results show that the leader is captured in 406 ms with a consumption of $1J$ to $9J$ per node for the linear network. And in case of a random network, it took 70 ms with an energy consumption of $1J$ to $5J$ per node. In these simulations, the energy required for a serialization of data from the microcontroller to the RF radio module is neglected. But, if we assume a serialization time of 38400 b/s then to find the leader requires 1.5 s and 190 ms for the linear and the random network, respectively. Determining an accurate estimator of the value of t_{max} in the case of random networks could be a topic for future work.

Fig. 1. A linear network with 100 sensor nodes.

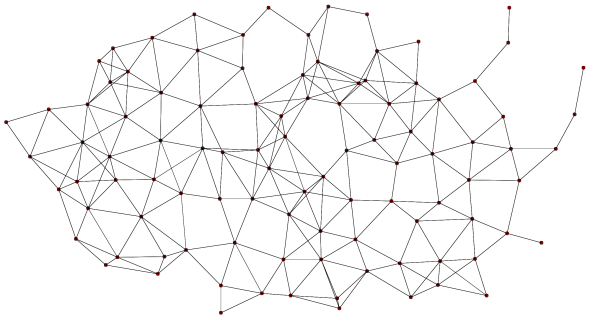

Fig. 2. A random network with 100 sensor nodes.

3 The Local Minima Finding algorithm

A local minimum node, also called *Local Leader*, is the node which has no neighbor with a value smaller than its own value. But this value is not necessarily a global minimum. The marked nodes, represented by the red arrows in Figure 4, show examples of local minima.

The Local Minima Finding (LMF) Algorithm uses the same principle as the previously presented *MinFind* algorithm to determine if a node is a local minimum or not, with the exception that each node will send its coordinates only once, and after receiving the messages from all its neighbors, it decides if it is a local minimum or not in case it receives a smaller value than its own. The algorithm of finding local minima is given as follows:

Algorithm 2 *LMF*: The pseudo-code of the Local Minima Finding Algorithm

Input: t, v
Output: local_min
1: local_min = true;
2: $x_{min} = v$;
3: send(x_{min}, *);
4: **while** ((x = read(t))$\neq$ *null*) and local_min) **do**
5: **if** ($x < x_{min}$) **then**
6: local_min = false;
7: **end if**
8: **end while**

4 The proposed method

4.1 Concept

In the *MinFind* algorithm each node is sending messages repeatedly and updates its values each time the received value is smaller than its own value. After a certain time, each node will be marked as a non-leader (or non-minimum) node, except the leader which has the smallest value since this node will never receive any smaller value than its own. This process is time-consuming and it requires a lot of broadcasting messages, which makes it very energy consuming and impractical in reality for the case of WSNs, because of collisions, for instance. To address this issue, we propose a new approach where each node will send a broadcast message once, in order to determine the local minima using the LMF algorithm (cf. Algorithm 2). Then each local minimum will send a message to a given reference node which will select the global minimum. This approach is detailed as follows:

Step 1: Mark each node as a global minimum and select one node as a reference node (cf. Figure 3).

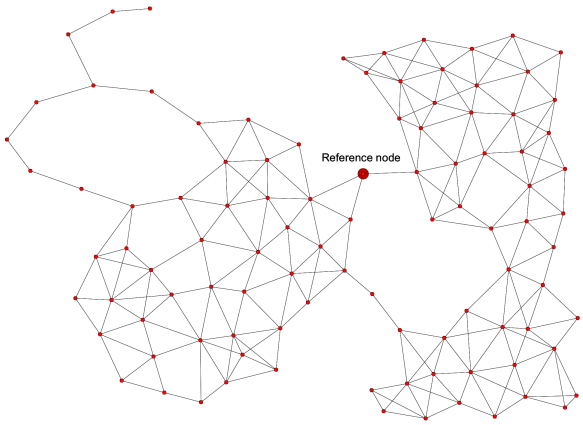

Fig. 3. Example of a network with a designed reference node.

Step 2: Run the LMF algorithm to find the local minima nodes and unmark the other nodes (cf. Figure 4).

Step 3: The reference node will send a message to the nodes in order to ask the local minima nodes to send their values (cf. Figure 5).

Step 4: Each local minimum node will send a message to the reference node and the reference node will determine the global minimum from the received local minima nodes (cf. Figure 6).

Step 5: The reference node will send a message to the global minimum node saying that it is the global minimum node (cf. Figure 7).

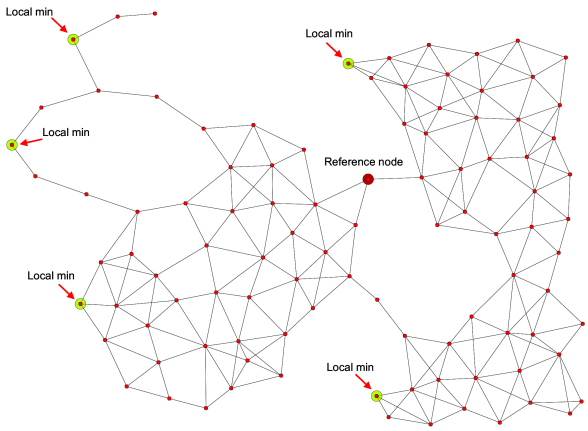

Fig. 4. The local minima found by Algorithm 2.

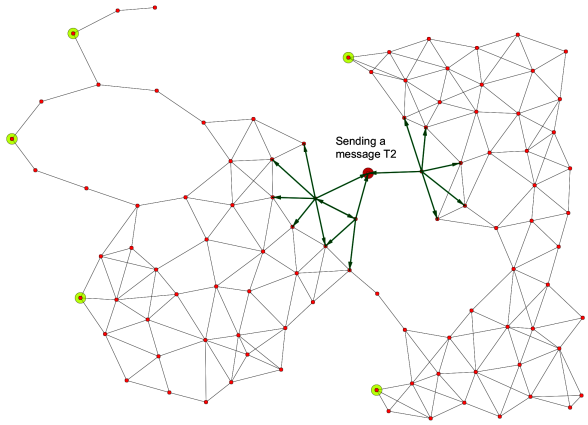

Fig. 5. The reference node asks for local minima nodes (flooding messages).

4.2 The LOGO Algorithms

To present the proposed algorithms, let us define in Table 1 some message primitives necessary for the communication between nodes and their definitions and in Table 2 the functions used in the algorithms. The proposed algorithm works in the case of bidirectional communication.

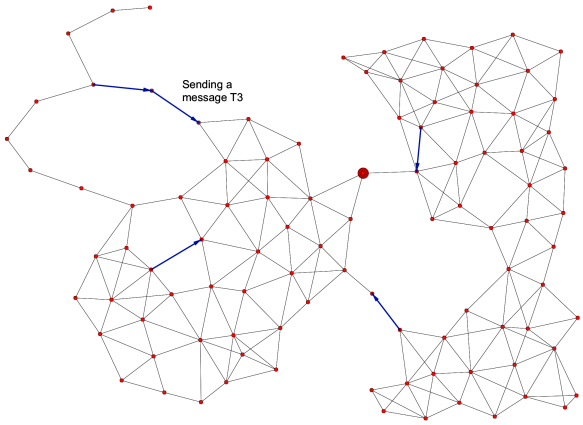

Fig. 6. The local minima nodes will declare themselves to the reference node (blue arrows).

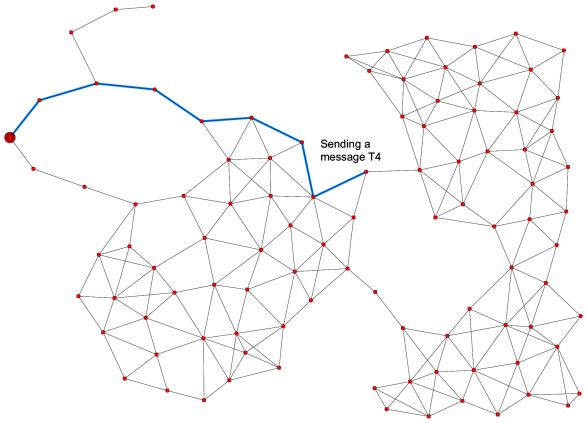

Fig. 7. The reference node designates the global minimum node (leader) and informs that node.

Table 1. Message primitives and their definitions.

Primitive	Definition
T1	I send you my value
T2	Send me your value if you are a local minimum
T3	I send you my value as a local minimum
T4	I want to inform the global minimum (leader election)

Table 2. Functions of the proposed algorithms.

Function	Definition
`getId()`	returns the node identifier
`delay(dt)`	waits *dt* milliseconds before going to the next instruction
`add(v,t)`	adds the value v at the top of the vector t
`pop(t)`	removes the value at the top of the vector t and returns it
`stop()`	stops the execution of the program
`send(a,b)`	sends the message a to the sensor node having the identifier b, or in a broadcast (if $b = *$)
`read()`	waiting for receipt of messages. This function is blocking, which means that if there is no received message any more, it remains blocked in this instruction
`read(wt)`	waiting for receipt of messages. If there is no received message after wt milliseconds then the execution will continue and go to the next instruction

Note that there are two algorithms. One is executed by the reference node (Algorithm 3) and the second is executed by the remaining nodes (Algorithm 4). Algorithm 3 of the reference node takes as inputs a value x and the time wt required before selecting the global minimum. The output *global_min* is a variable which is equal to true if the current reference node is a leader (minimum) and false, otherwise. It starts with an initialization (lines 1 to 3). And it waits for 1 second (line 4), the necessary time to finish the process of determining the local minima. This time must be changed if the number of neighbors of a sensor node is very important. It is the time required for any sensor node to send a message in a broadcast and receive mode from its neighbor nodes. Then it sends a message T2 to ask the local minima to send their value x (lines 5 and 6). In line 8, the reference node will wait for receiving a message containing the id of the transmitter (r_id), the value r_x of the local minimum message and t, the stack of the path from the local minimum node to the reference node. If a message is received before wt milliseconds, then it means that a message T3 is received from a local minimum node. In this case, the received value r_x is tested whether it is smaller than the current value x_min which at the beginning is equal to the local value x (line 18). If this is the case, the reference node will be declared as a non-global minimum (line 19), the value of id_min will be updated with the value of r_id (line 20), the value of x_min will be updated with the value of r_x (line 21) and the route t from the reference node to the local minimum (id_min) will be assigned to t_min (line 22). Otherwise, if the received message is null (line 9), which can happen when the node does not receive any message during the wt milliseconds, then the reference node has received messages from all the local minima. In this case, if the *global_min* value is equal to true, the reference node is the global minimum and the algorithm will stop (line 15). Otherwise, the route t is the one situated between the reference node and the global minimum

node. A message T4 will be sent to the global minimum, having the identifier id_min, using the route t (lines 11 to 13) in order to elect it.

Algorithm 3 *LOGO*: The pseudo-code of the reference node.

Input: x, wt
Output: *global_min*

```
 1: id = getId()
 2: x_min = x
 3: global_min = true
 4: delay(1000)
 5: message = (T2, id, null, null)
 6: send(message, *)
 7: while (true) do
 8:     (type, r_id, r_x, t)=read(wt)
 9:     if (type==null) then
10:         if (global_min==false) then
11:             n_id = pop(t_min)
12:             message = (T4, id_min, null, t_min)
13:             send(message, n_id)
14:         else
15:             stop()
16:         end if
17:     else
18:         if ((type==T3) and (r_x < x_min)) then
19:             global_min = false
20:             id_min = r_id
21:             x_min = r_x
22:             t_min = t
23:         end if
24:     end if
25: end while
```

Algorithm 4 of the remaining node takes as input only the value x. The output *global_min* is a variable which is equal to true if the current node is a leader (global minimum) and false, otherwise. Each non-reference node starts with initializations (lines 1 to 5). The variable once1 is used to allow only once the reception of T2 messages and the variable once2 is used to accept only once any received T4 message. Then it starts the process of the LMF by sending in a broadcast a T1 message in order to test if it is a local minimum or not by comparing the values received from its neighbors with its own value x. If any received value is smaller than its value, then the node will be considered as a non-global minimum (lines 9 to 15). Once all the values of the neighbors received, the algorithm goes to the second step, where it will wait for a message T2 initiated by the reference node. In this case, it will route this message to its neighbors and if it is a local minimum (*global_min* = true) then it will send the message T3 to answer the message T2 coming from the reference node, in

order to tell him that it is a local minimum (lines 22 and 23). Finally, it will be considered as a non-global minimum (line 24). The next part of the algorithm concerns the creation of the route from the local minimum node to the reference node. If any node receives a message T3 then it will add itself to a stack t (line 28) representing the route from the local minimum to the reference node, and route it again to the node p_id which had sent him previously a T2 message (lines 29 and 30). As soon as all the non-reference nodes have done this step, the reference node will be in the situation where he has received all the routes and values from the local minima and it chooses the one of the global minimum. Then it will send a message T4 to elect the global minimum (lines 11 to 13 of Algorithm 3). Finally, each non-reference node which receives a T4 message (line 32) containing the route t and the identifier r_id of the leader, will test if its identifier id matches the received identifier r_id (line 35). If yes, it will be elected (lines 36 and 37). Otherwise, it will route the same message to the next sensor node having the identifier n_id pulled from the route t (lines 39 to 41).

5 CupCarbon simulator and SenScript

The simulation of networks is an essential tool for testing protocols and their prior performance deployment. Researchers often use network simulators to test and validate proposed protocols and algorithms before their real deployment. Indeed, such an establishment may be costly and challenging, especially when talking about a large number of nodes distributed at a large scale. This is why the simulation of networks is essential. CupCarbon is a Smart City and Internet of Things Wireless Sensor Network (SCI-WSN) simulator. Its objective is to design, visualize, debug and validate distributed algorithms for monitoring, tracking, collecting environmental data, etc., and to create environmental scenarios such as fires, gas, mobiles, and generally within educational and scientific projects. It can help to visually explain the basic concepts of sensor networks and how they work; it may also support scientists to test their wireless topologies, protocols, etc., cf. Figure 8.

Networks can be designed and prototyped by an ergonomic and easy to use interface using the OpenStreetMap (OSM) framework to deploy sensors directly on the map. It includes a script called SenScript, which allows to program and configure each sensor node individually. The energy consumption can be calculated and displayed as a function of the simulated time. This allows to clarify the structure, feasibility and realistic implementation of a network before its real deployment. CupCarbon offers the possibility to simulate algorithms and scenarios in several steps. For example, there could be a step for determining the nodes of interest, followed by a step related to the nature of the communication between these nodes to perform a given task such as the detection of an event, and finally, a step describing the nature of the routing to the base station in case that an event is detected [18, 19].

SenScript is the script used to program sensor nodes of the CupCarbon simulator. It is a script where variables are not declared, but can be initialized. For

Algorithm 4 *LOGO*: The pseudo-code of the non-reference node.

Input: x
Output: *global_min*

```
 1: id = getId()
 2: x_min = x
 3: global_min = true
 4: once1 = false
 5: once2 = false
 6: message = (T1, id, x_min, null)
 7: send(message,*)
 8: while (true) do
 9:     (type, r_id, r_x, t) = read()
10:     if (type == T1) then
11:         if  (r_x < x_min) then
12:             x_min = r_x
13:             global_min = false
14:         end if
15:     end if
16:     if ((type == T2) and (once1 == false)) then
17:         once1 = true
18:         p_id = r_id
19:         message = (T2, id, null, null)
20:         send(message, *)
21:         if  (global_min == true) then
22:             add(id, t)
23:             message = (T3, r_id, x_min, t)
24:             send(message, p_id)
25:             global_min = false
26:         end if
27:     end if
28:     if (type == T3) then
29:         add(id, t)
30:         message = (T3, r_id, r_x, t)
31:         send(message, p_id)
32:     end if
33:     if ((type == T4) and (once2 == false)) then
34:         once2 = true
35:         if (r_id == id) then
36:             global_min = true
37:             stop()
38:         else
39:             n_id = pop(t)
40:             message = (T4, r_id, null, t)
41:             send(message, n_id)
42:         end if
43:     end if
44: end while
```

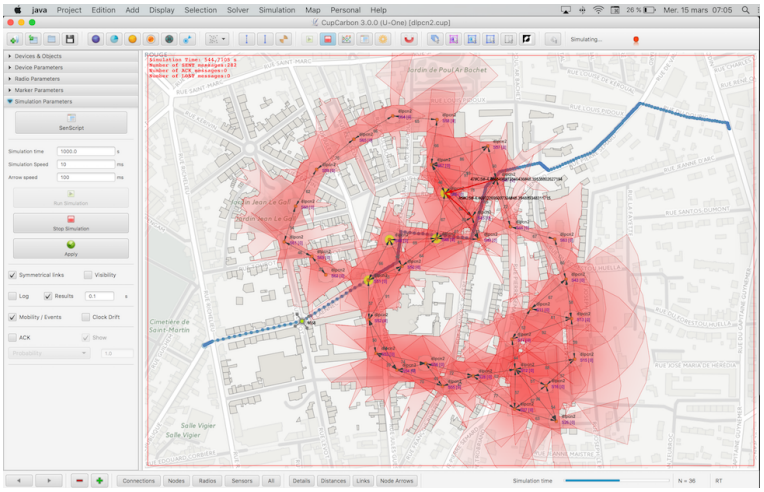

Fig. 8. CupCarbon User Interface.

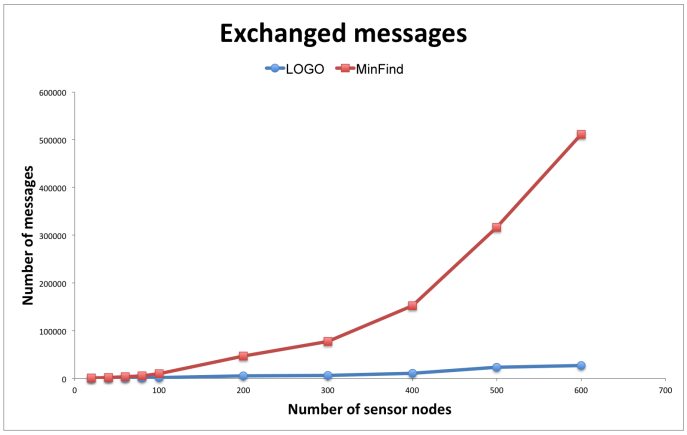

Fig. 9. Number of exchanged messages.

string variables, it is not necessary to use the quotes. A variable is used by its name, and its value is determined by $.

6 Simulation results

In this section, we will compare the proposed algorithm with the classical *MinFind* algorithm, since both of them can be used for any network. For the simulation, we have used the simulator CupCarbon [19], and SenScript is used to write the previously presented algorithms. We assume bidirectional communication

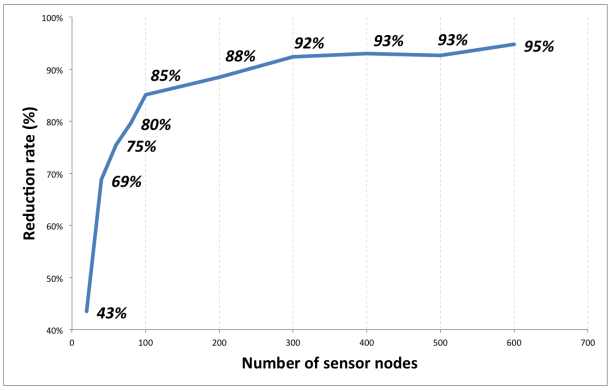

Fig. 10. Reduction rate in terms of the number of exchanged messages.

between nodes. Figure 3 shows an example of a wireless sensor network designed in CupCarbon. We have randomly generated 10 networks with 20, 40, 60, 80, 100, 200, 300, 400, 500 and 600 sensor nodes, respectively. For each network, we have calculated the number of transmitted and received messages (exchanged messages) in order to compare their energy consumption which is directly related to this metric. We have obtained the graphs of Figures 9 and 10. As we can see, the difference in each case can exceed 95% and this rate is increasing with the size of the networks. From this figure, one can conclude that for very large networks, this reduction can reach 99%.

7 Conclusion

We have presented a new Leader Election algorithm which is low energy consuming. This algorithm is called LOGO (Local Optima to Global Optimum) where sensor nodes that are local leaders will send a message to a reference node which will designate the global leader and elect it by sending it a selection message. The classical algorithm allowing to find this node is called Minimum Finding Algorithm. In this algorithm, each node sends its value in a broadcast mode each time a better value is received. This process is very energy consuming and not reliable since it may be subject to an important number of collisions and lost messages. Our proposed algorithm is more reliable since broadcast messages are sent only twice by each node, and the other communications are based on a direct sending. The obtained results show that the proposed algorithm reduces the energy consumption with rates that can exceed 95% compared to the classical algorithm. We are now working on comparing our algorithm with other methods and to implement it on real hardware sensor platforms. Combining the proposed idea with some computational intelligence technique as presented in [20] can also give more favorable results in energy consumption.

Acknowledgment

This project is supported by the French Agence Nationale de la Recherche ANR PERSEPTEUR - REF: ANR-14-CE24-0017.

References

1. Saoudi, M., Lalem, F., Bounceur, A., Euler, R., Kechadi, M. T., Laouid, A., Madani, B., and Sevaux, M., D-LPCN: A Distributed Least Polar-angle Connected Node Algorithm for Finding the Boundary of a Wireless Sensor Network, Ad Hoc Networks Journal, Elsevier, Volume 56, 1 March 2017, Pages 56-71.
2. Farid Lalem, Rahim Kacimi, Ahcène Bounceur, and Reinhardt Euler. Boundary node failure detection in wireless sensor networks. In IEEE International Symposium on Networks, Computers and Communications (ISNCC 2016), 11-13 May, Hammamet, Tunisia, 2016.
3. Santoro, N., Design and analysis of distributed algorithms, Vol. 56, John Wiley & Sons, 2007.
4. Sohail Jabbar, Abid Ali Minhas, Moneeb Gohar, Anand Paul, and Seungmin Rho, E-MCDA: Extended-Multilayer Cluster Designing Algorithm for Network Lifetime Improvement of Homogenous Wireless Sensor Networks, International Journal of Distributed Sensor Networks, Article ID 902581, ISSN: 1550-1329 (Print), ISSN: 1550-1477 (Online)
5. Sohail Jabbar, Abid Ali Minhas, Muhammad Imran, Shehzad Khalid and Kashif Saleem, Energy Efficient Strategy for Throughput Improvement in Wireless Sensor Networks, Sensors, Vol. 15, Issue 2, 2015, pp. 2473-2495,MDPI Publishers, 15, 2473-2495; doi:10.3390/s150202473
6. Beaulah Soundarabai, P., Thriveni, J., Venugopal, K. R., and Patnaik, L. M. An Improved Leader Election Algorithm for Distributed Systems. International Journal of Next-Generation Networks, 2013, 5(1), 21.
7. Effat Parvar, M., Yazdani, N., Effat Parvar, M., Dadlani, A., and Khonsari, A. Improved algorithms for leader election in distributed systems. In the 2nd IEEE International Conference on Computer engineering and technology (ICCET), 2010, Vol. 2, pp. 2-6.
8. Kim, T. W., Kim, E. H., Kim, J. K., and Kim, T. Y. A leader election algorithm in a distributed computing system. In Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems, 1995, pp. 481-485.
9. Chow, Y. C., Luo, K. C., and Newman-Wolfe, R. An optimal distributed algorithm for failure-driven leader election in bounded-degree networks. In IEEE Proceedings of the Third Workshop on Future Trends of Distributed Computing Systems, 1992, pp. 136-141.
10. Park, S. H., Kim, Y., and Hwang, J. S. An efficient algorithm for leader-election in synchronous distributed systems. In Proceedings of the IEEE Region 10 Conference TENCON, 1999, Vol. 2, pp. 1091-1094.
11. Brunekreef, J., Katoen, J. P., Koymans, R., and Mauw, S. (1996). Design and analysis of dynamic leader election protocols in broadcast networks, Distributed Computing, 9(4), 157-171.
12. Zargarnataj, M. New election algorithm based on assistant in distributed systems. In IEEE/ACS International Conference on Computer Systems and Applications, AICCSA'07, 2007, pp. 324-331.

13. Balhara, S., and Khanna, K., Leader Election Algorithms in Distributed Systems, Journal of Computer Science and Information Technology, IJCSMC, Vol. 3, Issue. 6, June 2014, pg.374-379.
14. Zargarnataj, M., New Election Algorithm based on Assistant in Distributed Systems, In IEEE/ACS International Conference on Computer Systems and Applications, Amman, 2007, pp. 324-331.
15. Singh, G. Efficient distributed algorithms for leader election in complete networks. In 11th IEEE International Conference on Distributed Computing Systems, 1991, pp. 472-479.
16. Lynch, N. A., Distributed algorithms, Morgan Kaufmann, 1996.
17. Garcia-Molina, H., Elections in a Distributed Computing System, IEEE Transactions on Computers, Vol. C-31, No. 1, 1982, pp. 48-59.
18. CupCarbon simulator, http://www.cupcarbon.com
19. Mehdi, K., Lounis, M., Bounceur, A., and Kechadi, T. CupCarbon: A Multi-Agent and Discrete Event Wireless Sensor Network Design and Simulation Tool, In IEEE 7th International Conference on Simulation Tools and Techniques (SIMUTools'14), Lisbon, Portugal, 2014.
20. Sohail Jabbar, Rabia Iram, Abid Ali Minhas, Imran Shafi, Shehzad Khalid, Muqeet Ahmad Intelligent optimization of energy aware routing in wireless sensor network through bio-inspired computing: survey and future directions, International Journal of Distributed Sensor Networks, Vol. 2013, Article Id 421084, 13 pages, 2013